Homomorphic Encryption
EDA284 - Literature Study

JOHN CROFT and ANNA-MARIA UNTERBERGER

Additional Key Words and Phrases: homomorphic encryption, cryptography, computer architecture and organization, fast fourier transform, FFT, NTT, security

ACM Reference Format:
John Croft and Anna-Maria Unterberger. 2024. Homomorphic Encryption EDA284 - Literature Study. 1, 1 (February 2024), 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Encryption encodes information (plaintext) in such a way that third parties can not read it (ciphertext). Only the sender and recipient are able to access the original message. To do this, additional information in the form of keys is required. The ciphertext will depend on the key used.

There are two key-based encryption schemes: symmetric and asymmetric. In symmetric key schemes, the same key is used to both encrypt and decrypt a given message. Asymmetric key schemes use a *public* key to encrypt the message and a *private* key to then decrypt it.

In the current era of cloud computing and distributed databases, data storage and processing is being outsourced to third parties. With conventional encryption schemes, the data must be decrypted, processed and re-encrypted whenever it is to be modified. Privacy is thus compromised at this point and there is no way to guarantee that data is not leaked by the third party.

Homomorphic encryption (HE) is an encryption scheme in which encrypted data can be processed without the third party ever being aware of the actual content. HE extends the basic idea of key-based cryptography with the concept of an *evaluation* capability that allows computations to be performed directly on ciphertext without the need to decrypt it. Any *evaluation* will produce a new ciphertext, and decrypting it will produce a result corresponding to data as if the operations had been performed on it in plaintext. Thus, in order to outsource computation with HE, it is enough that the server has an encrypted version of the data and that the client merely sends the desired function for manipulating the data. As a side note, computations are also called *circuits* in HE literature, as all desired computations (or functions) can be represented by boolean or arithmetic circuits. Despite the name, these can be of arbitrary depth and complexity.

The practical use of HE has the potential to revolutionise certain applications in which data is too sensitive to be allowed to leak, but is desired to be shared or outsourced for processing. This is particularly true in highly regulated industries where privacy concerns are paramount, such as online banking.

Authors' address: John Croft, croft@student.chalmers.se; Anna-Maria Unterberger, annunt@student.chalmers.se.

The main computational challenge of HE is the fact that the primitive functions (described later in the paper) utilize multiplication and addition with operands on the order of millions of bits. This is inefficient and time-consuming on most platforms.

2 BACKGROUND ON HOMOMORPHIC ENCRYPTION

Conventional cryptographic systems usually contain the following three algorithms:

- KeyGen(λ) : A key **k**, used to transform plaintext into ciphertext and vice versa, is generated. λ is a security parameter that will determine the security strength of the key and therefore the cipher.
- Encrypt(k, m) : The message **m** is encrypted by using the encryption key **k**. The output is the ciphertext **c**.
- Decrypt(k, c) : The ciphertext **c** is transformed back into plaintext by applying the decryption key **k**.

HE schemes extend the traditional encryption approach by adding the function Evaluate (Eval):

- Eval(C, $(c_1, ..., c_n)$, k): By using the key **k**, the circuit (or desired computation) **C** is applied to the ciphertexts $(c_1, ..., c_n)$ and returns the produced ciphertext **c**.

An encryption scheme is called homomorphic over an operation $\star$ if it supports the following equation: $E(m_1) \star E(m_2) = E(m_1 \star m_2), \forall m_1, m_2 \in M$, where E is the encryption algorithm and M is the set of all possible messages [1].

Homomorphic Encryption schemes can be classified into three categories [1]:

- Partially homomorphic encryption (PHE) can only perform one type of operation (either addition or multiplication) on ciphertext, but an unlimited number of times. Arbitrary computations can therefore not be performed, limiting its practical use to algorithms that can be implemented with only one of these operations. One example of a suitable application is e-voting, which in its most basic form only requires the tallying of votes (ie. addition) but where privacy concerns can not be overstated.
- *Somewhat* homomorphic encryption (SWHE) can perform several types of operations, enabling arbitrary computations, but only a limited number of times since the the ciphertext grows with each homomorphic operation, and beyond some threshold the data can not be recovered anymore. Beyond the obvious limit imposed by a monotonically growing ciphertext, it is the notion of cryptographic *noise*, introduced by this scheme, that causes the data to degrade to the point that it can no longer be discerned.
- Fully homomorphic encryption (FHE) allows arbitrary computations on ciphertext an unlimited number of times. With the help of so-called *bootstrapping* (to 'refresh' the ciphertext and reduce *noise* levels) and *squashing* (to reduce the requirements needed for the bootstrapping phase) techniques, the limit on number of operations is effectively removed, and any SWHE scheme can thus be converted into an FHE scheme.

A breakthrough in the field, headed by Craig Gentry in his seminal PhD thesis, suggested a feasible technique to implement FHE using ideal lattices [6], though the computational requirements were later shown to be astronomical in an implemented system and the performance was typically on the order of 1 second per processed bit [7]. The key size was also on the order of gigabytes; unacceptable for practical use, in other words.

The main barriers (in terms of computational requirements) of FHE is the *bootstrapping* operation required, which involves evaluating its own decryption algorithm *circuit*, as well as the cost of homomorphic multiplications. It is the latter that is the main target for potential speedup in dedicated hardware solutions.

3 SCOPE

In this project we will focus on FHE schemes. Other schemes such as SWHE have already been shown to be fast enough to potentially be practical [1], if limited in application (see earlier drawbacks of SWHE). FHE is still prohibitively expensive in terms of efficiency [1] and thus benefits most from any potential speedup that can be wrought from architecture-level optimisations.

Several FHE implementations have been developed after Gentry's first proposal of a lattice-based FHE scheme in 2009. Even though his idea was very promising, it proved to be inefficient in terms of its computational costs and thus applicability in real life [1]. Multiple other schemes have emerged since then with better efficiencies as the most computationally expensive parts of the procedure are addressed.

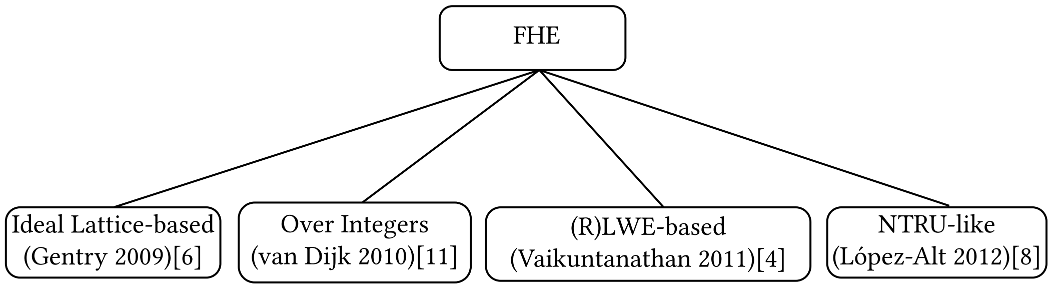
FHE
Ideal Lattice-based (Gentry 2009)[6]
Over Integers (van Dijk 2010)[11]
(R)LWE-based (Vaikuntanathan 2011)[4]
NTRU-like (López-Alt 2012)[8]

Fig. 1. The Main FHE Families

In terms of software, we will focus on the (R)LWE[1]-based system, since all publicly available HE libraries are based on it. Furthermore, we will utilise HElib, which is a software library that implements the Brakerski-Gentry-Vaikuntanathan (BGV) homomorphic encryption scheme [3], as a reference.

4 ALGORITHM DESCRIPTION AND EVALUATION

5 EXISTING COMPUTER ARCHITECTURES

There are several different approaches in order to make FHE more efficient using different hardware platforms. Each has their own tradeoffs in terms of cost, performance, flexibility, complexity and energy-efficiency. This section discusses the different approaches in a high-level manner.

5.1 Performance of FHE on a general purpose CPU

Acar et al. [1] conducted a survey on the theory and implementation of software-based HE schemes, tested on commodity hardware.

Advantages: General purpose systems are cheap and easily accessible. Furthermore, they are the most flexible in terms of programmability. In contrast with GPUs and FPGAs, algorithms will not need special tailoring to run on regular processors (unless one wishes to leverage special CPU features, such as SIMD instructions). Multiple software libraries that are aimed at regular CPUs are available.

[1](Ring) Learning With Erros

Disadvantages: The inherent generality of CPUs makes them unsuited to highly specialized workloads such as this. Clever changes to algorithms can be made to attempt to improve performance, but CPUs can generally not compete with hardware that is more inherently suited to the problem (ie. massive parallellism, arithmetic on very long operands etc...)

Conclusion: They do not provide a feasible alternative for FHE computations.

5.2 FPGA Implementations

Field Programmable Gate Arrays (FPGAs) are a great asset for the secure and efficient implementation of cryptographic algorithms due to their potential to accelerate a wide range of applications [10]. Cilardo et al. [5] designed and implemented an FPGA-based accelerator to speed up large integer multiplication. They utilise the Schönhag-Strasse algorithm [9] which provides an advantage for operands in the order of 100.000 bits [5].

Advantages: Arbitrary functions can be implemented in hardware. Cryptographic schemes regularly have peculiarities (ie. bit-level manipulation) that may be less supported on CPU/GPUs. FPGAs usually also contain dedicated and optimised multiply-accumulate (MAC) units that can be utilised when multiplying large operands, a common bottleneck in FHE schemes. FPGA implementations focusing on the FFT multiplier block have been shown to be twice as fast as a competitive GPU-based one and significantly faster than software-only implementations on general purpose CPUs [5].

Disadvantages: FPGAs have relatively high per-unit cost and overall energy consumption, as well as relatively low clock speed when compared to ASICs and GPUs. This comes as a consequence of the hardware needed to support the reconfigurability. Also, it may be impractical to fully implement FHE. Rather, FPGA implementations in literature target key bottleneck areas and are intended for use in conjunction with more conventional computers. Furthermore, hardware designs are generally not beholden to any standard, making interfacing potentially challenging.

Conclusion:

5.3 Utilising a GPU

Another approach to accelerate FHE is the use of a graphics processing unit (GPU) for that purpose. Wang et al. [12] based their work on the Gentry-Halevi FHE scheme [11] by computing the costly million-bit modular multiplication in two steps:
1. To efficiently calculate large number multiplication, the Schönhage-Strassen algorithm [9] is used to recursively calculate FFT. The computations utilise the massive parallelism on a GPU
2. Barrett reduction is applied to perform modulo reduction [2].

Advantages: FHE schemes are highly parallelisable with respect to data, which has the potential to take advantage of the inherently parallel operation of GPUs.

Disadvantages: Algorithms must be tailored to make use of GPU architecture features and to avoid unsupported operations such as recursion[12].

Conclusion: The GPU implementation described in [12] yielded a speedup of 7X for the FHE primitives (encryption, recryption and decryption) over the reference CPU-based implementation described in [7]. This is as significant improvement, but still too inefficient to be deployable in commercial applications.

REFERENCES

[1] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. 2018. A Survey on Homomorphic Encryption Schemes: Theory and Implementation. *ACM Comput. Surv.* 51, 4, Article Article 79 (July 2018), 35 pages. https://doi.org/acar

[2] Paul Barrett. 1987. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Advances in Cryptology — CRYPTO' 86*, Andrew M. Odlyzko (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 311–323.

[3] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2011. Fully Homomorphic Encryption without Bootstrapping. Cryptology ePrint Archive, Report 2011/277. https://eprint.iacr.org/2011/277.

[4] Zvika Brakerski and Vinod Vaikuntanathan. 2011. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In *Advances in Cryptology – CRYPTO 2011*, Phillip Rogaway (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 505–524.

[5] A. Cilardo and D. Argenziano. 2016. Securing the cloud with reconfigurable computing: An FPGA accelerator for homomorphic encryption. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE).* 1622–1627.

[6] Craig Gentry. 2009. *A fully homomorphic encryption scheme.* Ph.D. Dissertation. Stanford University. crypto.stanford.edu/craig.

[7] Craig Gentry and Shai Halevi. 2010. Implementing Gentry's Fully-Homomorphic Encryption Scheme. Cryptology ePrint Archive, Report 2010/520. https://eprint.iacr.org/2010/520.

[8] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. 2012. On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing (STOC '12).* Association for Computing Machinery, New York, NY, USA, 1219–1234. https://doi.org/10.1145/2213977.2214086

[9] A. Schönhage and V. Strassen. 1971. Schnelle Multiplikation großer Zahlen. In *Computing 7.* Springer Berlin Heidelberg, Berlin, Heidelberg, 281–292.

[10] François-Xavier Standaert. 2009. *Secure and Efficient Implementation of Symmetric Encryption Schemes using FPGAs.* Springer US, Boston, MA, 295–320. https://doi.org/10.1007/978-0-387-71817-0_11

[11] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. 2010. Fully Homomorphic Encryption over the Integers. In *Advances in Cryptology – EUROCRYPT 2010*, Henri Gilbert (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 24–43.

[12] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar. 2012. Accelerating fully homomorphic encryption using GPU. In *2012 IEEE Conference on High Performance Extreme Computing.* 1–5. https://doi.org/10.1109/HPEC.2012.6408660